A PERSONNEL DATA BASE RETRIEVAL SYSTEM

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

Major R. K. SHARMA

to the

DEPARTMENT OF COMPUTER SCIENCE

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JULY 1977

CS-1977-M-SHA-PER

CERTIFICATE

CERTIFIED that the thesis entitled, 'A PERSONNEL DATA BASE RETRIEVAL SYSTEM' has been written by Major R.K. Sharma for the work done under my supervision and this has not been submitted elsewhere for a degree.

July 8, 1977

R. Sankar

Dr. R. Sankar
Professor of Computer Science
INDIAN INSTITUTE OF TECHNOLOGY
KANPUR.

ACKNOWLEDGEMENTS

I wish to express my deep sense of appreciation and gratitude to my thesis supervisor, Professor R. SANKAR for his continuous guidance and help at all stages of my work.

I am very much thankful to my colleagues, Lt. Col. R.K. BAGGA, DEEPAK GHANEKAR and AJOY K. MUKHERJEE for their valuable discussions and help during the execution of the project.

I thank Mr. H.K. NATHANI for his help in typing this thesis.

Kanpur
July 8, 1977

- Major R.K. Sharma

CONTENTS

ABSTRACT

A personnel data base consists of integrated data about persons of any organization. A retrieval system retrieves information from this data base. The information may be either of a single individual or about a group of people. This system has been developed to retrieve information from a relational data base containing data about Army Officers. The computer system chosen for this data base is TDC-316. The system is based on disk. This project was taken up as a part of data base project group. The other parts being

(a) Building a Data Base

(b) Data validation

All the systems are written in the Basic Assembly Language (BAL) of the TDC-316.

1. INTRODUCTION

This information retrieval system works on a personnel data base developed at IIT Kanpur as a project on TDC-316 computer. A personnel data base has data about persons. In this case the persons chosen are army officers. For management of army officers, there is a requirement of storing and retrieving a large amount of data in an efficient manner. The old technique of storing data in multiple files, each suitable for particular application is a cumbersom and inefficient technique. The modern data base with its management techniques is an ideal choice for working on such data. Out of the type of data bases available, a relational data base offers the best scheme for flexibility and working in an integrated fashion. This particular data base and retrieval system is based on the relational approach.

There are large number of terms being used in data bases. The terms and their meanings, as applicable in this study are given in the succeeding paragraphs. The terms definitions are generally as per Computer Data Base Organization by MARTIN and from DATE.

Data Base

A data base may be defined as a collection of interrelated data stored together with as little redundancy as possible to serve one or more applications in an optimal fashion. The data are stored so that they are independent of programs which use the data. A common and controlled approach is used in adding new data and in modifying and retrieving existing data within the data base.

An integrated data base contains data for many applications. It is a repository of information needed for running some organization. In our case the organization is the army. A data base may be organized for batch processing, real time processing or on-line processing where single transaction is completed one at a time, without the constraints of real time processing. Our system approximates to INLINE processing.

Advantages of having data bases against the conventional tape library systems have been explained by various authors. The biggest advantage is having controlled minimal redundancy with its accompaniment of

low cost storage
single updating run
consistancy.

One of the most important characteristics of data bases is the case of restructuring data storage without change in application programs. This is termed Data Independence.

<u>Data Independence</u>

It implies that data and the programs which use this data, are mutually independent and each may be changed without affecting the other. The application programmer is insulated from changes in data storage. He need not know how and where they are stored. This also gives some measure of security and privacy. Unanticipated questions may be asked from the data base. The application programs are interested in logical organization.

<u>Logical and Physical View</u>

A logical data description refers to the manner in which a programmer sees the data. It may or may not (more often not) coincide with a physical data description which refers to the manner in which data is stored physically on devices. Programmer sees the logical relationships and logical data structures. He requires a logical chain of records, which may not be the physical chain. Software converts programmers logical statements to the physical reality.

<u>SCHEMA</u>

The logical description of a data base used by the data base software is called SCHEMA. A schema is a chart of the types of data that are used. It gives the names of the entities and their attributes and specifies the relations between them. It is a frame work into which the values of data items can be fitted. The values may change from time to time. Any particular set of values in the schema is called an 'Instance of Schema'. Schemas are often drawn in the form of block diagrams. They are also called 'Data Model'.

The following terms which have been used in defining SCHEMA are themselves defined.

<u>Data Item</u>

A data item is the smallest unit of named data. It is referred to as a <u>field</u> or <u>data element</u>.

<u>Data Aggregates</u>

It is a collection of data items within a record which is given a name and referred to as a whole, e.g., <u>Data</u> may consist of year, month and day.

<u>Record</u>

A named collection of data items or data aggregates. An application program reads a logical record.

<u>File</u>

A file is a named collection of all occurrences of a given type of record.

<u>Entity</u>

Items about which we store information are referred to as entities. An entity is a tangible object such as a person, a part of inventory or a place. It may be non-tangible such as an event, a customer account. We record the properties of the entities. The information is called an entity record. It is called a logical record by application programme and a stored record by the physical storing mechanism.

Attributes

The property of an entity is called an attribute. These attributes have values such as personal number, rank, address, account number etc. These attributes are stored as data items in byte strings.

FLAT FILE

The correlation between values, attributes and enties is normally most easily stored in a fixed sequence in two dimensional tabular form, called a flat file. A scheme of flat file is shown in the Figure 1-1.

Each row of the table represents one entity. Each column of the table refers to an attribute. The name of the attribute is given in the top of the table. An attribute the value of which identifies an entity uniquely is called an identifier or key. In the case shown, personal number acts as the key. The grouping of data items represents a relationship between these items and such a related set of values is also called a tuple.

Subschema

A subschema is an application programmer's view of the data he uses. Many subschemas can be derived from a single schema. The application programmers do not necessarily see the schema. With the subschema, the schema and the physical organization, there are three view of data.

A FLAT FILE

	Attribute 1	Attribute 2	Attribute 3	Attribute 4
Key	Personal No.	Name	Date of birth	Address
Entity 1	IC-001	Ravi	12 Dec. 1940	xx
2	IC-610	Madhav	15 Sept. 1938	xx
3	IC-200	Deepak	8 Marc. 1948	xx
4	IC-300	Ram	1 Apr. 1950	xx
5	IC-400	Raji	2 Jun. 1956	xx

Fig. 1-1

(1) Subschema : application programmer's view also called Data Sub Model.

(2) Schema or Global logical data base description as seen by the data base administrator. Also called DATA MODEL.

(3) Physical Data Base Description: A chart of the physical layout of the data on storage devices seen by systems programmers.

The system software provides mapping between these different views. Thus a data base management system will have a presentation as shown in Figure 1-2. (MARTIN Figure 7-2.)

This system ensures logical data independence which means that the overall logical structure of the data may be changed without changing the application programs.

It also has physical data independence which means that the physical storage may be changed without changing either the overall logical structure of the data or the application programs.

Relational Data Base

Once the logical data independence has been achieved the over riding consideration in design of data base is the convenience of the majority of application programmers. The description of data should be understood easily by the users, such as a description is achieved by using two dimensional flat tables. Any complex representation such as

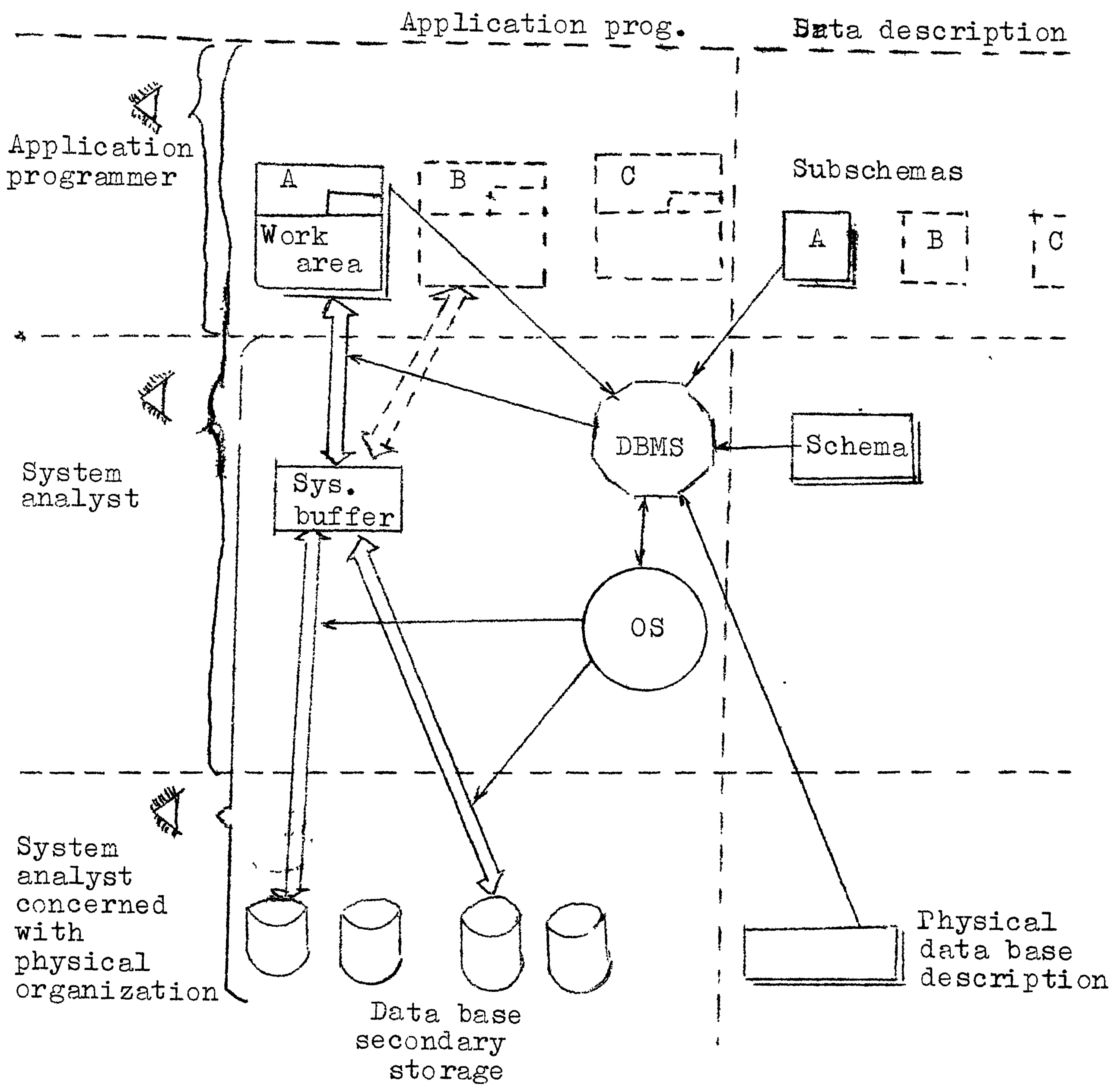

Application prog.
Data description
Application programmer
A
Work area
B
C
Subschemas
A
B
C
System analyst
Sys. buffer
DBMS
Schema
OS
System analyst concerned with physical organization
Data base secondary storage
Physical data base description

Figure 1-2.

tree or plex structures can also be reduced to a flat file representation by a simplification process called normalization. A flat file tabular representation is called a relation.. A data base constructed using relations is called a relational data base.

It should be possible to extract subsets from such a table and also to join such subsets giving a flexibility to the user in information retrieval. This retrieval will consist of selecting different columns for an entity and the ability to combine them. The characteristics of such a data base will be

1. Each entity in a relation represents one data item.
2. The relations are column homogeneous i.e., all items in a column are of same type.
3. Each column or attribute has a distinct name.
4. Each row is distinct. Duplicate rows are not allowed.

Information Retrieval

Information retrieval from such a base can be done in any sequence of rows and columns. Each entity has one identifier attribute which is unique. In some relation there may be more than one attribute to identify the entity. These identifiers are also called keys. These keys have two properties:

(a) unique identifieation

(b) non-redundancy.

Where there are more than one keys, one of them is termed as primary key. Most of the search is done on primary key. In our system, an approach akin to relational calculus is adopted. The user simply defines the results he wants and leaves the system to decide what operations are necessary to extract that information. It is like defining a new relation which is to be derived from the existing relations in the base. Sometimes it is advocated that binary relations (relations having binary tuples) offer the maximum flexibility for random queries in an unanticipated fashion. This however increases the house keeping jobs of the system software. Our data base is not based on binary relation, however, a large number of queries of unanticipated types can be answered.

Army Officers Data Base

The army has a large number of officers. Maximum possible information on each has to be kept. At present about 150 different attributes can be easily identified. The users are the various departments of the army which are interested in reports as below.

(a) strength of officers in the army

(b) strength of officers in the various corps

(c) pay and accounts

(d) qualification in [illegible]

(e) pensions

(f) recruitment through various academies

(g) selection of officers for various courses and appointments.

Most of the army officers movements are done corpswise As such it is beneficial to have a relation which is organized with corps as primary key. But this key will be repeated this type of file is called an inverted file. Other relations are organized on personal number as the primary key. Every officer has a unique personal number. The user departments give their subschemas, which consist of two or more attributes. The system extracts this information from the data base. The system developed here does not do the pay and accounting though this can also be done with slight modifications. This information system prepares lists of officers as out put. For manipulation and calculations, simple programs have to be written. The system is a pilot system for a personnel data base developed specifically on TDC-316. Due to limitation of time and the availability of computer simplicity in design of programs has been the key. The system can be easily expanded to suit any unanticipated query requirement. This thesis has been organised manner:

Part I: System Constraints and General Considerations.

Part II: Description of Relations and Common Data Structures.

Part III: Query Translation.

Part IV: Access Mechanism.

Part V: Processing and Output.

Program listing and instructions for working the system have been included as appendices.

2. SYSTEM CONSTRAINTS

Introduction

The retrieval system starts from the projection of the query. The query has to be recognized, analysed and then used to define data structures which are used by seek and processing routines. The information is then located, processed and put in the output buffer. The output routines then give a print out as per the subschema of the user.

Such a system can be as powerful as desired with a very wide range of features. These again are controlled by the following factors:

(a) The computer used

(b) The language used

(c) The memory available

(d) Complexity of the system

(e) System speed

(f) Generality of the system

(g) Time available to generate the system.

These factors as applicable to this system are described in the following paragraphs.

Computer

The computer to be used is TDC-316 with the following configuration:

1. CPU
2. Memory 28K
3. Card reader
4. Disc Pack - 7.6 M bytes
5. Printer
6. Paper tape reader
7. Paper tape punch
8. Key board/teletype.

This configuration can have only card reader/keyboard/paper tape as the input device and line printer/paper tape punch as output device. The system is disc based.

Language

There are two languages available:

1. FORTRAN
2. Basic assembly language, BAL.

The BAL assembler is not relocatable and as such FORTRAN and BAL routines cannot be called by each other. FORTRAN has an added restriction in use of disc. It can access the disc only in the serial mode. It was decided to use BAL as the language for system development.

Memory Available

The main memory available with BAL is 28K words. Each word in TDC-316 consists of 2 bytes of 8 bits each. Each byte is seperably addressable. The lower byte has been even address. For word addressing the location counter has to be incremented by 2. The secondary memory available is a disc pack of 7.62 M bytes. There are 202 usable tracks/cylinders, 10 surfaces and 10 sectors/surface/track. Each sector of disc can store 256 bytes. Every sector is addressable through a surface and sector register word in the disc controller.

Complexity of System

The complexity of the system is judged by the number and type of software routines, then lengths and the facilities provided. The levels of nesting of subroutines, generality and sophistication of input/output requirements are other factors.

This system is totally modular in construction. The routines are kept small in size and designed for optimum efficiency in working time and space. The number of pointer and flags have been kept to a minimum

System Speed

The algorithms have been designed to exploit all the features of the assembler to reduce working time and space. The movement of data between the memory and the disk is kept at the bare minimum. This has been achieved by avoiding creation of intermediary files on disc while processing conditions.

Example

Let us say the set of conditions is as per the table given below.

Sr.No.	Relation	Field (attribute)	Condition	Value
1	R1	F1 (RANK)	EQUAL	CAPT.
2	R1	F4 (ARM)	L.T.	Arty
3	R1	F6 (Service)	G.T.	5 years
4	R2	F2 (Qual)	EQUAL	B.Sc.
5	R2	F8 (POINTS)	GTE	4
6	R3	F15 (Age)	L.T.	26 years

Table 2-1. Set of Conditions

The conditions show that we have to process 3 Relations R1,R2,R3. We can do either of the following

(a) Search relation R1 completely. Create an intermediary file containing list of officers meeting conditions 1,2,3 pertaining to Relation 1. Then we process Relation R2 and extract another intermediary file II. Which will have a list of all officers satisfying conditions 1 to 5. Finally we obtain the final list or file after Relation 3. In this case two intermediary files will have to be created on disc.

(b) The second approach will be to process relations R1,R2,R3 simultaneously. Processing of every officers record is through a filter made up of JOIN or logical AND of all the conditions. An output record is created only when all six conditions are satisfied. No intermediary files are created.

In the second approach, movement between the memory and disc is cut down. However, space is required for processing more than one relation in the memory. It will be seen than in the absence of inverted files, no extra space is required. For every inverted file, an extra buffer is required and pointers have to be kept for the last record processed in every relation. This design uses the second approach.

Generality of the System

The algorithm have been designed in a most general fashion to suit any personnel data base. To achieve maximum efficiency for this application. Certain data structures have limited size and width. For example, condition tables has a variable field width of upto 8 bytes, i.e., fields of lengths upto 8 bytes can be used for search criteria. These sizes can be varied easily to suit any other data base applications. The dialogue between the user and the computer may have to be varied slightly.

Time Availability

Time available for generation of this systen has been limited by the M.Tech. program length. Many more features can be added to the existing facilities. Some suggestions are made in the last chapter.

-

3. ORGANIZATION OF RELATIONS AND DIRECTORIES

This chapter deals with the organization of data and the data structures that help to locate the physical address of the desired information.

Layout of Relations

A relation consists of a set of similar records. A record consists of two or more fields which are interrelated. These fields are placed adjacent to each other in a record. Records are placed adjacent to each other in memory. A tabular representation of a relation is shown in Figure 3.1.

Field A	Field B	Field C	
CAPT.	123	10	RECORD 1
LT.	122	6	RECORD 2
MAJ.	124	7	RECORD 3

In memory the representation will be

RECORD 1	RECORD 2	MAJ.	124	7

Figure 3-1. Layout of a Relation.

Key

Each of these relation has a particular field as primary key for identifying records. Some relations use more than one keys especially inverted files. A primary key is used to

RELATIONS DIRECTORY

	Relation Name	Relation call	Pointer to FDLIST in memory	Length of FD list	Pointer to Index in memory	No. of entries in INDEF	Total No. of records
Name	RELNM	RELID	PTRFDL	NOFLD	AMINX	NIND	TO REC
Size	6	2	1	1	2	1	2
Example	GRPS	01	40,000	4	60,000	20	6,000

	Current Record No.	Primary key field	Format of primary key	Secondary key field	Format
Name	CRCNO	KEYRP	FMKRP	KEYRS	FNKRS
SIZE	2	2	2	2	2
Example	204	07	3	05	8

.o locate a group of records and a secondarykey used to locate a particular record out of that group. A relation of this type in this data base is RELATION ARM which uses the field CORPS to identify all officers belonging to a particular corps of service and then uses the officers personnel number to select a unique record.

Relation Directory

The logical organization of relations is cdefined by a data structure named Relations Directory. Its layout is given in the table listed on the opposite page. This directory gives all the particulars needed for processing. Its components are described below:

1. RECNM gives the relation name. It may consist of upto 6 characters.
2. RELATION CODE Each relation is given a code number. In RERIL our case it happens to be the serial number in the directory.
3. POINTER TO LIST OF FIELDS It gives the address of the PTRFDC list of fields which form this relation.
4. LENGTH OF FIELD LIST It gives the number of fields in NOFD that relation.
5. AMINX It gives the location of the relations index table in memory. This INDEX table is a physical layout of the relation.

6.	NIND	It gives the number of entries in the primary index table. It shows the number of disc cylinder being used by that relation.
7.	TOTREC	Gives the total number of records.
8.	CRCNO	Gives the current record number.
9.	KEYRI	Gives the primary key of that relation.
10.	FMKRI	Gives the length of the primary key.
11.	KEYRS	Gives the secondary key.
12.	FMKRS	Gives the length of the secondary key.

Field Directory

This directory is maintained to store the information about the fields used by the system. The layout of this directory is given below.

Field Code	Address of Relations list	Name	Length

Field List

This list is maintained to co-relate relations fields. It has the following format

Field Identification No.	FORMAT	FDIST

This field list structure is used by the query interpretation part to fill up other data structures which are necessary for processing the query. Its description is as follows.

Field Director

Field Code	Address of relations list	Name	Length

Field List

FD ID		FORMAT		F Dist
FD ID	Relation ID	Length	Type	Distance from beginning of
2 Bytes		2 Bytes		1 Byte

Figure 3-3

FDID

It is a two byte field. First byte gives the Relation Number and the second byte gives the field number.

FOMAT

It is two byte field. First byte gives the length of the field and the second byte gives the type of field whether alphanumeric or numeric.

FDIST

It gives the distance in bytes of this field from the beginning of the record.

The routines take the required information from these tables and construct other data structures for their own use in SEEK, SEARCH and PROCESS. These will be described in the following chapters.

-

4. MAIN SCHEME AND DATA STRUCTURES

Objectives

The objectives of the personnel data base retrieval system may be of the following types.

(a) Find out some or all details of a particular officer.

(b) Find out some or all details of a set of officers who have the attributes as specified by the user.

The first type is termed as SINGLE QUERY. The second type is termed as GROUP QUERY. Examples of each will be given later under query translation. Both the types of queries are quite common.

Requirements for tackling the Problem

In both the type of queries the job breaks down to

(a) Tabulate what fields are the condition fields which have to be matched for some values. Also tabulate information about their location. This is tabulated in TBPROC.

(b) Tabulate what fields are to be given as output. These are tabulated in TBOTPT.

(c) Tabulate the conditions, their values and fields which have to be matched. COND TABLE.

(d) SUBSCHEMA of the user in final output format.

Action Steps

First of all the data structures have to be built up by query translation. Once these data structures are built up, one relation is selected and the following steps taken

[a] Seek the physical location of the sector desired.

[b] Search the sector to locate likely record.

[c] Scan the TBPROC for finding fields to be matched.

[d] Compare these field values of the record against the conditional values.

[e] If they agree, scan the TBOTPT for fields to be output, extract them from the record and make an output buffer.

[f] Go and repeat Steps [a] to [e] for other relations mentioned in TBPROC.

[g] Reject the first output at any time, any of the conditions do not match as COND TABLE signifies logical AND of all conditions.

[h] Once all the relations are over the final output buffer is reorganized as per sub-schema and put out on the printer.

The data structures used are described below:

Data Structures

The following data structures are described

(a)	TBPROC	Table for processing
(b)	TBOTPT	Table for output combined with TBPROC
(c)	COND	Table for conditions.
(d)	SBCHMA	Subschema of the user.

TBPROC/TBOTPT

This table stores fields which are to be processed or output along with necessary information about their length and layout of the relation. The information in the buffer will be of the following types.

Fields	1	2	3	4	1	2	3	4	1	2	3	4	1
	Record 1				Record 2				Record 3				Rec 4

This particular relation has a record format consisting of 4 fields of varying lengths

FD1	length l_1 bytes
FD2	length l_2 bytes
FD3	length l_3 bytes
FD4	length l_4 bytes

A query may require only fields 1 and 4 to be processed and F2 to be output. Similary other relations will have other fields. The tabulation will be as follows:

TBPROC	Rel.	Fd.
	R1	F1
	R1	F4
	R2	F5
	R3	F6
	R3	F10

TBOTPT	Rel.	FD.
	R1	F2
	R2	F5
	R3	F10

To extract these relations we must also have the details about the field lengths and locations of the fields within the relations. These two columns are added as FDLGT (Field length) and FDIST (field distance from beginning of record)

Since this information is required for both TBPROC and TBOTPT both these tables have been merged into one table with an extra column added which stores information about the treatment of the field e.g., whether this field has only to be output. Layout of the table is as given on the facing page.

The treatment of the field takes the following shape.

DIGIT	ACTION
Store 0	Field has to be output only.
Store 1	Field has to be processed and OUTPUT.
Store 2	Field has to be processed only.

The width of each entry is 5 bytes. The last entry will be zeroes in all columns to signify end of all fields.

TB Proc/TBOTPT

Relation	Field	FD Length	FD Dist	Treatment
R_1	F_1	l_1	0	2
R_1	F_2	l_2	d_1	0
R_1	F_4	l_4	d_2	1
l_2	F_5	l_5	d_5	1
l_2	F_6	l_6	d_6	0
R_3	F_8	l_8	d_8	0
0	0	0	0	0
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

Figure 4-1. Table for Processing and Output

Treatment

2 = Process only
1 = Process and output
0 = Output only

CONDITION TABLE

	FD ID	FD Value	condition
Length	2 Bytes	8 Bytes	2 Bytes
	01	C-12345	Equal
	X	XXX	NEQ
	X	XXX	LE
	X	XXX	GE
	X	XXX	GT
	X	XXX	LT
	X	XXX	ANY

Equal	
NEQ	Not equal
LE	Less than or equal to
GE	Greater than or equal to
GT	Greater than
LT	Less than
ANY	All values ok

Figure 4-2.

COND TABLE

The fields which have to be compared have their values and the type of comparison stored in this table. The structure is shown in Table No .

Its entry length is 12 bytes. The first two bytes give the field identification. The next eight bytes are for the value and the last two bytes are for storing the type of comparison.

During seek and process this table is used for matching values. The following type of comparisons are specifically allowed.

(a) Greater than (GT)

(b) Greater than or Equal (GE)

(c) EQUAL

(d) Not Equal (NE)

(e) Less than (LT)

(f) Less than or equal (LE)

(g) ANY

This table is designed for a more maxl lengths of 8 bytes. In this application names are excluded from comparison. The routine which scans this table is COMP.

SBCHMA

This data structures holds the subschema given by the user. Its structure is as follows:

FD Identity	Type of FIELD
F1	Numeric
F3	Alpha
F4	Numeric

Table 4-1. Subschema Data Structure.

This structure is also created at the time of query translation and is used at the output time. The numeric fields will have to be converted into byte strings in a printable mode. The Alphanumeric information does not need conversion and can be directly given as output. The conversion of numeric fields from byte strings into binary number equivalent is done to save storage space and save comparisons.

These are the data structures used most often during processing. There are two other structures which do a mapping between logical address and physical address of records and sectors. These will be described in the chapter on the SEEK. From here we go onto the actual mechanism starting from the first step, The Query Translation.

-

5. QUERY TRANSLATION

<u>USER Definition</u>

A Query has ~~to~~ two parts:

(a) set of conditions which have to be satisfied.

(b) Set of fields which form the output in a particular order. This lay out is also called the subschema.

Both these sets may be mutually exclusive i.e., have no fields in common, or may not be mutually exclusive. The user has to define both these sets.

<u>Query Translation</u>

Systems have been developed which facilitate the user in defining the conditions and subschema. They then extract proper information and develop data structures which are used for processing the information. Query translation then means filling up of the required data structures by software routines.

<u>Example</u>: A typical example in this case will be as follows.

<u>Subschema</u> - output

FDS # RANK, # RANK, # NAME, # DATE OF BIRTH

of the officer with

<u>condition</u>

FD# PERSONAL No. = 1000.

This forms a <u>single</u> officer query. The Group Query may table the shape.

<u>Subschema</u>

FDS. #RANK, #NAME, #DATE OF BIRTH

of all officers with

<u>conditions</u>

FD # Personal No. less than IC-2000.

FD# Parent Arm. equals # ARMOUR.

This is a group query.

<u>Difference between a Group Query and a Single Query</u>

A single officer will have only one condition listed, his personall number. It may have any number of fields in the subschema.

A group query will have more than one conditions listed. The personal number is always assumed as a condition as it forms the necessary link between different relations.

<u>Dialogue Between the User and the System</u>

An interactive system which permits the user to have a running dialogue while forming the query is very desirable. In the system designed here the user is in full conversation with the system, till his query is fully translated. All the fields in subschema and the conditions are displayed before him on the keyboard.

<u>Choice of Input Media</u>

This system enables the user to define his query either through the keyboard or the card reader. He is asked to exercise his choice of input device.

Security

A code check is incorporated to authenticate the user. The user has to declare the code prevalent at that time. The DBA can change this code at his will and unauthorised user is debarred.

Description of the System

Function: The system is called APPROACH. Its basic function is to obtain the subschema and the set of conditions required for seek and process routines.

Data Structures Used: These are

(a) COND TABLE

(b) TBPROC/TBOTPT

(c) SBSCHMA

(d) FD. DIRECTORYFDLIST.

The COND TABLE will list the conditions specified, the TBPROC/TBOTPT will list the fields to be processed/output. SBSCHMA will hold the subschema, and FD DIRECTORY is used to obtain various informations about the fields specified. These structures have already been defined.

Mechanism: In the first part the system starts a dialogue with the user to establish his bonafide authority to use the system by asking him the code. Failure to supply this code rings a bell continuously. Other security measures can be suitably linked. Then the system asks him whether he wants retrieval/update/building of data base. On being given the correct answer (letter R) retrieval is allowed to begin.

TBPROC

The second part begins with establishing the type of query, whether SINGLE or GROUP. This w ill decide the processing. It also gives him a choice of input device, card reader/keyboard. The user is then asked to list the fields which will have to be processed or output with that specification. With his specification the data structure TBPROC is built up. It is then printed out as a table on the keyboard to enable the user to verify.

COND. TABLE

The third part asks for the conditions for processing. A set of fields with values to be matched is obtained from the user. Distinction is made between a SINGLE QUERY and GROUP QUERY. The numeric fields are converted from the keyboard byte string into the binary form by subroutine ENCODE. The condition/inequalities are put into COND table by subroutine CONDIT and the COND table is completed for use.

SUBSCHEMA

The fourth part obtains the user subschema and builds SBCHCMA data structure to be stored and used for the final output. It is also printed out at the keyboard. After build up of these structures, the routine APPROACH passes on to SEEK routines for further processing of queries. It also signals the end of user interaction.

Subroutines Used

This routine makes use of the following subroutines

(a) SCAN

(b) NOWR

(c) CONDIT

(d) ENCODE

Output

(a) TBPROC

(b) CONDTABLE

(c) SBCHMA.

6. ACCESS MECHANISM

This chapter deals with the SEEK routines and data structures used by them to physically locate the sector which contains the desired information and bring it into the memory for processing. It has to find the track number, the surface number and the sector number of the disc.

<u>Sequence</u>

The sequence of access is -

(a) select a relation from the query

(b) scan the relation directory for the address of index tables, keylengths by routine FNDKEY.

(c) scan the primary index table to obtain the track/cylinder number by routine TRKDS.

(d) Bring the secondary index table i.e., the sector map of the track from the disc to memory.

(e) scan this table to obtain the surface and sector number by FNCSEC.

(f) bring the desired sector into memory by READ.

(g) mark the index tables for future reference.

The routines mentioned in the above paragraph i.e., FNDKEY, TRKDS, FNCSEC, READ, are assisted by two other routines.

(i) COMP   which scans the COND table.

(ii) FNDIX   forms a part of both TRKDS and FNCSRC.

Physical Layout of Relations

The relations are stored on the disk. Each relation may have one or more tracks allotted to it. Every relation starts from a fresh disk. The TDC disk has 202 tracks (usable) each having 10 surfaces. Each surface has ten sectors for every track. A sector can store 256 bytes. On the surface 0 of every track the first few sectors are utilized for storing the layout table for that track. The method of seeking a sector is described below.

Physical Mapping

Indexed sequential method is used for storing and locating the records. Two tables are kept as mapping devices -

(a) Primary Index Table

(b) Secondary Index Table

Primary Index Table

This table locates the track number. There is one such table for every relation. The location of these tables are given in the relations directory. These labels are stored in main memeory. Its layout is given in the figure 6.1.

	Highest key in track	Track No.	Length of secondary index table
SIZE	Variable	1 byte	2 bytes
Example	IC-001234	67	80

Table 6-1. Primary Index Table

This table is scanned by subroutine TRKDS.

<u>Secondary Index Table</u>

This table helps to locate the surface and sector number. There is one such table for every track. This table is located on surface 0 of every track. Its length names as per the key length. Its layout is as below:

No. of Records in the Sector	Highest Key No.	Surface No.	Sector No.	
2 bytes	Variable	1 byte	1 byte	Length
51 7	4568	0	0	Example
51		0	1	
⋮		⋮	⋮	
25		2	5	

Table 6-2. Secondary Index Table.

Since there are 100 sectors on every track, there will be a maximum of 100 entries in a table. The length of the table may vary from 1 sector to 4 to 5 sectors of 0 surface, depending upon the key length and total number of entries.

With the help of these structures a sector is brought into memory. This sector then has to be examined record by record in a serial fashion by process routines described later ~~below~~. The working of the SEEK routines is described in the following paragraph.

INDEX TABLES

Primary Index Table

	Highest Key in track	Track No.	Length of secondary index
Size	Variable	1 Byte	2 Byte
Example	IC-200	50	80

Secondary Index Table

	No. of records in sector	Highest key in sector	Surface	Sector
Length	2 Bytes	Variable	1 Byte	1 Byte
Example	51	0	0	0
	51	0	0	0
	⋮	⋮	⋮	⋮
	25	IC-300	2	5

Figure 6-1

Subroutine FNDKEY

Object: To locate primary key, primary key length, memory location of primary index table.

Data Structure used: Relations Directory

Output: PRKEY, PKYLT, RLOC, SRKEY, SKYLT, ~~IXENTR~~ PXENT.

External symbols: ROFST, KEYRP, KEYRS, FMKRP, FNKRS, AMINX, NIND, TEMP, MQ, ZERO, MESG1.

Working:

The relation directory is organized columnwise. The relation number is taken as the offset from the beginning of columns and the proper value stored in output variables.

PRKEY	contains primary key field identity
PKYLT	contains primary key length
SRKEY	contains secondary key field
SKYLT	contains secondary key length
RLOC	contains location of the primary index table.

Subroutine TRKDS

Object: To find out the cylinder address and load it into the disk control word.

Data Structure Used: Primary index table and COND table.

Output:

TKDR	Track address word
IXPTR	Primary index pointer
TOTENT	Total number of records in the secondary index table.

External Symbols Used: CFLOP, PKYLT, TEMP, IRKEY, RLOC, PKPTR, FNDIX, TKDR, MESGZ.

Flow Chart: It is given in the facing page.

Working: This subroutine puts the length of primary index table in variable TEMP and calls subroutine FNDIX to scan the table for the highest key entry which matches the key given in condition lable. Then it reads the track address from the table and puts it into variable TKDR. It also puts the length of secondary index table for that track in TOTENT.

Seeking the Surface and Sector

Subroutine FNCSEC

This subroutine finds out the surface and sector number in which the information is lying.

Data Structure Used

Secondary Index Table.

Organization

On Every track, the surface 0 contains the index table. Since a track comprising of 100 sectors, there will be 100 entries in the index. If all the sectors are not used, the end of index table will be unaltered by placing a special word in the NREC (No. of Records) field of the Index Table

Working

1. First put the length of entry into X.
2. Read the surface 0, Sector 1, into Buffer BUFF.
3. Get the No. of Records in the index sectors from the first word of the BUFFER.

4. Set pointer SXPTR to point at BUFF+2.
5. Test whether this byte is zero. If it is zero, add the displacement x to it and go to next entry.
6. If it is not zero, then SXPTR will now have address of the first data entry.
7. Calculate various variables from this location as reference.

At the end of this part of subroutine the following variables will have proper values.

(a) SXPTR

(b) LNREC

(c) CTR3

(d) NISEC

(e) NREC (No. of records in that secto

Now go to the other part i.e., to FINDIX. This subroutine searches for the proper key field value. The exit from the subroutine can be either of the two.

(a) Success: the information has been found. Put SFDOI=0

(b) Failure: the information has not been found. Put SFDOI=1.

<u>SFDOI = 1</u> Search Unsuccessful

If the search is unsuccessful, the following action is taken

(a) Increment sector counter. Compare its value with the NISEC (No. of sectors used for Index Table.)

(b) If the sector counter is > NISEC, there is some error the information has not been found in the index. Write a message to that effect and stop.

(c) Sector counter = NISEC. This is the last sector put LNREC (No. of records in the last index sector) in NREC increment the sector No. in surface sector word (SSUR) for disk controller and read the next sector.

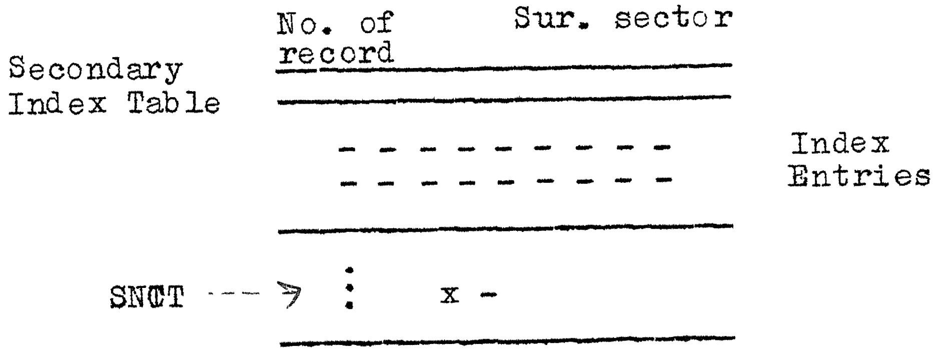
(d) Sector Counter < NISEC. The sector number is [illegible] SSUR and go to read the next sector.

<u>SFLOP=0</u> <u>SEARCH FROM INDEX SUCCESSFUL</u>

Take the following action:

(a) Transfer the contents of sector and surface address in the SSUR for disc control.

(b) Transfer pointer to number of records in this sector to SNCT. This will now store the location of the wanted record entry in the index table.



(c) Store the current surface and sector address in SAVSEC.

(d) Clear all variables not required.

(c) Return from the subroutine.

END

At the end of the subroutine, the surface and sector address is stored in the SSUR and also SAVESEC. All other variables retain the values already mentioned except for CTR3 and TEMP and sector counter.

SEEK Mechanism Finish

Once the seek mechanism has gone through subroutines TRKLS and FNCSEC, the track address and the sector surface addresses are put into the disc controller and the system is ready to read the ~~corrector~~ correct sector for the required information. Control passes to main routine.

SUBROUTINE COMP

Object: In the seek phase of retrieval, this subroutine is used by FINDEX subroutine. It scans condition table for the key field value to be compared with the value in the index (either primary lable or secondary lable) tables, to locate the correct track, surface and sector number.

DATA structure used

1. COND TABLE.

Calling Instruction

```
1. JMS   R5, COMP
   ARG ≠ 1   Word containing primary key fied (PRKEY)
   ARG ≠ 2   Word containing primary key length (PKYLT)
   ARG ≠ 3   Pointer to location of first byte of field
             value to be compared.
```

Symbols Used: CRT2, INEQTR, CODE, VALCOM, FCOND, FDLGTH, VALLOC, AKY, MATCH.

Output: MATCH = 1, Means comparison success

MATCH = 0, means comparison unsuccessful

Working: Initialize:

This subroutine makes use of Register 3 and 4. It also initializes counter CER2 which keeps a track of bytes compare

Locating the FDID in COND Table

L1: The subroutine puts the absolute address of COND into R3. It transfer the 1st argument i.e., the FD IDENTITY to CODE.

L2: It then checks whether there are any conditions set up in condition table by comparing contents of FDID in COND table with zero. If it is zero, i.e., there is no condition set, the subroutine branches to signal success. Puts MATCH=1 and goes out.

L3: If the first entry is not zero, it compares contents of code with the FDID in card table. If it matches the subroutine branches to compare that FD value.

L4: If it does not match, then the length of the cond table entry is added to the R3 to point at the next FDID entry and branches to L3. It also keeps a count of the number of times this search takes place. If it exceeds the capacity of cond table without finding that particular FD, it signals an error. FD code not found and stops. It will signal this condition also if it encounters a zero in the FDID field of COND table. That signifies end of condition tables.

FD CODE IDENTIFIED   BLOCK MAT

Setting of Symbols

Once the field code has been identified, the algorithm has to compare the FD value transferred from the calling routine with the FD value in condition table under the condition mentioned in the COND table. It starts transferring remaining arguments i.e., the length of the primary key with FDLGTM and loc. of the FD VALUE to be processed in VALLOC. It stores the starting point of COND table FD value in VALCOM. It sets the location of INEQUALITY in FCOND, R3 and R4 are set to point at the beginning of values to be compared in the condition table data buffer respectively.

Check Whather SEEK or PROCESSING Phase   Block MALL

If the call has come from the processing phase, the condition which has to be tested now will be listed in FCOND, i.e., GT or LT, LTE, NEQ etc. On the other hand if the call has come from SEEK Phase then the only condition which may be checked is whether the FD value tested in the INDEX table is GTE to the value in the COND table. Since the values listed in the Index tables are the highest key values in that track or the sector, i.e., primary index table. This is sufficient for a single type of query where all conditions will be put to EQUAL.

PN a group query, the condition may be LE. In that case we must start processing from Track 1. Similarly if the inequality is GT we should test for greater than, not greater than or equal to.

This logic is done by a group of instructions which first check AKY. AKY is an indicator for SEEK operation if it is set

P1: If it is not 1, then go to the condition tested in the COND table.

P2: If it is 1, then check condition table for EQUAL. If it is EQUAL go to GTE part.

P3: If check condition is not EQUAL then go to the condition listed in the COND table.

Value Comparison

Once the field has been identified, pointer set to point to field value location, the control jumps to the condition testing blocks as per the condition listed in the COND table these can be:

1. EQUAL	=
2. LT	Less than
3. GT	Greater than
4. GTE	Greater than or equal to
5. LTE	Less than or equal to
6. NEQ	Not equal
7. ANY	Any condition.

The algorithm compares the field values byte by byte, the comparison is in the shape of (R4-R3), i.e., value to be processed - value to be checked in COND table. There is counter CTR2 to keep a count of bytes compared. Any time a condition is satisfied, the program jumps to either FAIL or SUCCESS.

<u>Success/Failure</u>

After every byte is compared, the counter is incremented, when it becomes equal to FDLGTM (Field length given earlier), it jumps to fail or success. This happens when all the earlier bytes have been equal and the last byte is the deciding factor.

SUCCESS: If the comparison has been successful MATCH is made equal to 1, A message~~s~~ is given to console and subroutine returns.

FAIL: If the comparison fails MATCH = 0. Subroutine passes a message and returns. In either case Register R3, R4 are restored.

<u>Logic of Comparison</u>

EQUAL (1) Compare one byte of both R4, R3. If they are not equal, go to FAIL. If they are equal GO TO FEQL1.

(2) <u>FEQLI Block</u>: increments counter, compares it with FDLGTN, if it is less than FDLGTM, increment R3 and R4 and go back to Step 1.

(3) If counter is GTE ~~equal~~ to FDLGTM, that means, it has finished comparing all the bytes since it has come from equal it is a successful comparison, i.e., CTR $\geq$ FDLGTM - GO TO SUCCESS.

LT (1) Compare one byte of R4, R3.

(2) If R4 $<$ R3, i.e., sign bit is set, branch to success.

(3) If R4 = R3 i.e., zero bit is set, branch to FEQL.

(4) If R4 $>$ R3 sign bit is 0. Go to FAIL.

FEQL block does the reverse of FEQL1. If counter is equal to FDLGTM, that means the two quantities are equal and LT is not satisfied, GO TO FAIL.

Going to FEQL and FEQL1 means the earlier comparison was equal. If it is the last byte, then the whole of the values were equal.

GT (1) Compare one byte.

(2) If equal go to FEQL

(3) If R4 $>$ R3 branch to success

(4) If R4 $<$ R3 branch to FAIL.

GTE (1) Compare one byte of R4 and R3.

(2) If R4 $>$ R3 success

(3) If R4 = R3 branch to FEQL1.

(4) If R4 $<$ R3 - Go to FAIL.

LTE (1) Compare one byte of R4 with R3

(2) If R4 $<$ R3 - success

(3) If R4 = R3 - FEQL1

(4) If R4 $>$ R3 FAIL.

NEQ (1) Compare one byte

(2) If they are R4 $\neq$ R3 (zero is clear) Branch to Success.

(3) If they are equal R4 = R3, branch to FEQL.

ANY BRN to Success.

SUBROUTINE FNDIX

Object: This subroutine scans the primary index table and the secondary index tables when called for by subroutines TRKDS and FNCSEC, and gives out the location of the required KEY FD value in the index table.

Data Structures Used

(1) Primary index table

(2) Secondary Index table

(3) COND table.

Subroutines used: COMP.

Symbols Used: CTR1, CTR3, LOC, MESG9, FRKEY, FKYLT, MATCH, CFLOP, NREC, SFLOP, MESG16, MESG10, X, AKY-FLAG

Calling Instruction: JMS R3, FNDIX
ARG1 (word loc)

Working

Initialization Transfer ARG: Block A1

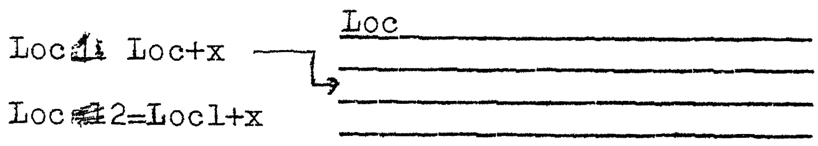
It increments counter CTR3 which keeps a count of Records processed in the secondary index table in subroutine FNCSEC. It also transfer the pointer to the FL value in LOC. In addition it sets a flag AKY = 1 to indicate that it is a SEEK operation (used in COMP subroutine). It then checks whether there is any data in LOC. If it is zero that signifies the end of the Index table and is an error condition. Print a message.

Scan of Table

(1) The pointer LOC points ~~t~~to the location of 1st byte of the FD value to be compared. The FNDIX routine calls subroutine COMP to compare the values. The result is either MATCH = 1 or MATCH = 0.

(2) If MATCH = 1, then the search is over and branch to success block SURSEC and return. The value LOC will contain the proper location of entry desired.

(3) If MATCH = 0, then we have to scan the next line of index table. Advance location by the length of the entry of the index table (X). Block A2.



(4) After it advances the pointer, the algorithm goes back to step A1, to increment counters and checking for a zero in the first byte of the FD value.

FINDIX in secondary table search

If the table being traversed is secondary table, then there are some additional checks to be made. As such after return from comparison with a Match value = 0. It establishes whether it is secondary ~~index~~ lable or not by checking the CFLOI flag. This flag CFLOI is reset by the TRKDS subroutine. It is set to 1 by the FNCSEC routine. In the end of query this flag should be cleared.

Action for Secondary INDEX Table (Block SECY)

Since the secondary table resides on the disc, only one sector is brought into the Memory [illegible]. At this time NREC contains the number of records in the sector. As the checking proceeds on the information, counter 1 CTR1 keeps a track of records processed. If this counter becomes equal to number of records, i.e., IF CTR1 = NREC that mean all the records in the sector have been processed with no match found. It then sets a flop SFLOI to 1 to signify failure as far as this sector is concerned and returns to the subroutine ENCSEC, which will bring in a new sector of the index table and recommence search.

Success

Once the desired entry has been located, the subroutine prints an exit FNDIX message and returns.

Errors

If no match is found it prints a message INDEX TABLE OVER and stops.

7. PROCESSING SINGLE OFFICER QUERY

The search for information can be of the following types:

(a) search for particulars of one officer

(b) search for a block of officers

(c) search an entire file.

The factors involved in all these types of searches are discussed in this and the following chapters.

Individual Officer Search

In this type of search, one individual key like the personal number of an officer is given and one or more other fields information is desired. An example -

OUTPUT (a) Date of birth

(b) Year of commission

(c) Qualifications

(d) Next to kin

for Personal No. IC-2000, Capt. S. SINGH.

The fields on which information is desired may be from one relation or may span more than one relation. The personal number being the key for all relations, except for the inverted files, all such relations can be scanned. The method of search employed will be the standard procedure for search based on primary key. This method is outlined below:

Method of Search and Process

1. SEEK

(a) Select the first relation

(b) Scan the primary index table and the secondary index tables to locate the physical sector on the disk.

(c) Bring the sector to the main memory buffer.

2. Process

(a) Search this buffer on the primary key and locate the particular record.

(b) Extract the fields desired to be put out and put them in output buffer.

(c) Check whether all fields required have been obtained, if yes, go to (e). If not go to (d).

(d) Obtain the next relation and go to back to 1(b)

(e) Arrange the output in the subschema format (Data sub model) and print.

(f) Go to end Routine.

In this type of processing sectors of the current relation being processed can overlay the buffer of the previous relation. The change over of relations require no reassignment of pointer/ buffer. The routines used in this type of query processing are:

(a) SEEK

(i) FNDKEY
(ii) FNCSEC
(iii) FINDIX
(iv) COMP
(v) TRKDS

(b) PROCESS

(i) PROC
(ii) OTPT.

SEEK

The method of finding out the physical sector/block and how to bring it into memory has been explained earlier. Once the information is in buffer it has to be processed as detailed below.

SEARCHIN-Buffer

The location of the record in the buffer has to be done by a sequential search. Provision also has to be made for skipping a blank or deleted record, and errors signals to be provided if identification of record is not obtained. Such steps are out lined below.

S1: Read a record

S2: Check primary key field

S3: If it is a blank or key value does not match the key given go to S4. Else go to S5.

S4: Is it end of sector or end of file? If yes signal error else advance pointer and go to S1.

S5: Check fields required by the OTPT table and put them in the buffer.

S6: Arrange fields as per subschema.

Information in Tables

COND

In this type of retrieval, there will be only entry in the COND table, that of personal number, EQUAL Value

01	VALUE	EQUAL

This value will be checked every time a record has to be checked.

Process and OTPT Tables

In a general case the TBPROC has to be scanned to check all the fields which have to be compared in every relation. A pointer is positioned at the head of buffer. Field are selected from the TBPROC and successively compared with values in buffer records are processed till a record is found which matches all the field conditions of one relation. Then PROC takes on the next relation and so on.

In the single query, the above method is not referred. to. The only field to be compared is Primary key field. If we were to follow the general method for every relation to be processed the personal number will have a corresponding entry in TBPROC. The structure of the TBPROC/TBTPT will be as in the table given below:

TBPROC

REL. NO.	FDID	FD LENGTH	FDIST	ACTION
R1	01	8	0	1
R1	03	4	8	0
R2	01	8	0	~~1~~ 0
R2	05	4	12	0
R3	01	8	0	0
R3	07	10	12	0

We see here that FD 01 which is the key field for all relations comes in for comparison in every relation. Also care will have to be taken to see that it is not repeated in the OTPT of every relation. This complication is avoida-ble for single query and the modification is described below.

<u>Modification</u>

We know that the primary key field is the first field of every relation. The distance from the beginning of every record FDIST = 0. Its field identity is known as well as its field length.

Once we have this information in memory, we put FDIST=0, bypass the TBPROC scan and start comparing the value in the buffer to with that given in the COND table, When a match is struck pass on to OTPT routines.

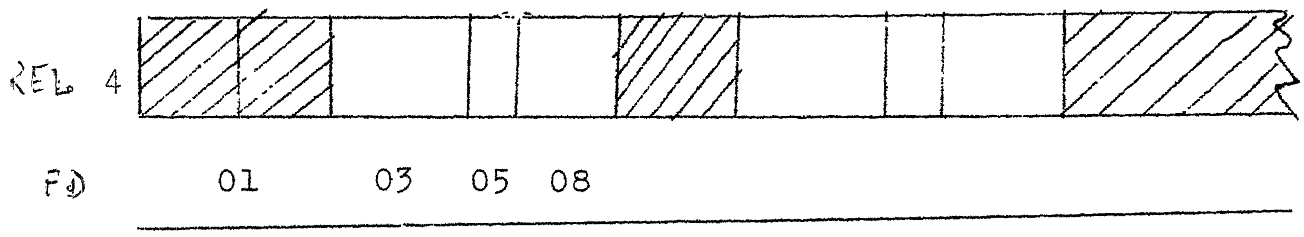
For this query, the TBPROL/TBOTPT may not contain the key field entry at all unless this key is also desired as the output field.

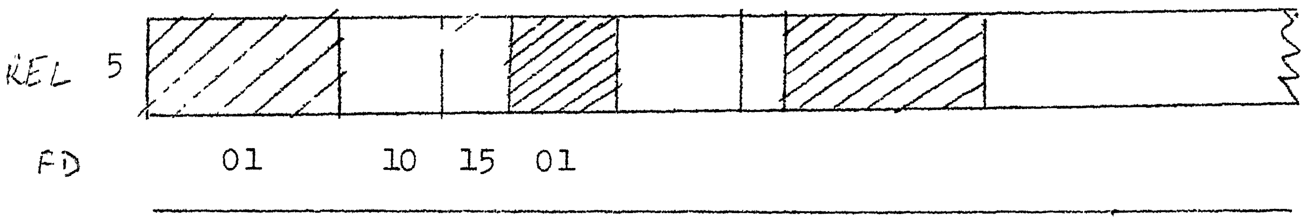
<u>Stopping the Algorithm</u>

This algorithm deals with only one particular record per relation. Once this record has been located, all fields output, the system should stop and give a message job over. If the information is not found in that sector, then there is some error and an error message should come. Thus the algorithm should stop in the following states.

(a) JOBOVER after OTPT routines over.

PROCESSING SINGLE QUERY





Layout TBPROC

REL ID	FD ID	FD LGT	FD DIST	Action
R4	1	8	0	1
R4	5	2	12	0
R4	8	4	14	0
R5	10	4	8	0
R5	0	0	0	0

Details

REL	FD	Length LGT
4	01	8
	03	4
	05	2
	08	4

REL	FD	Length LD6
5	01	8
	10	4
	15	2

Figure 7-1.

2. <u>Error in Sector</u>

If it is first relation and the officers record is not found, there is error in SEEK. Give error message.

<u>Checks To Be Made</u>

Once the SEEK is over, the only checks to be made while processing are:

(a) Is it a blank record. If yes go to next record.

(b) Is it end of sector.

(c) Is it end of file.

These checks have to be made every time a record is processed. In case it is case (b) or (c) and it is not relation 1, the output should be filled with blanks. In case of Rel. 1, signal error.

<u>METHOD OF PROCESSING</u>

The method of processing is illustrated by an example. Take a case in which infomation is to be obtained from two relations Rel 4 and REL 5. The layout of these relations and the structure of TBPROC is given on the facing page. The first relation to be processed is REL 4.

<u>SEEK</u>

The seek mechanism puts a sector of REL4 in the memory buffer BUFF. [illegible] symbol. The seek mechanism also gives the following information

1. NREC contains no. of records in that sector

2. RECLGT contains records length

3. RELID contains relation ID.

PROC

The processing algorithm first transfers this information to the variables mentioned (Block T ARG).

It then checks for type of query by TST SINGLE. In SINGLE case i.e., SINGLE = 1, FDIST = 0, FDLGT = 8. This information is put into two variables, KFDLGT, KYFD and transferred also to variables FDLOC, FDLGT. Now the portion of processing TBPROC is bypassed. A pointer BUFFTR is positioned at the start of BUFFER in the beginning and it keeps an incrementing by RECLGT to keep pointing at the head of successive records. Another pointer TEMP is utilized to point to the location of the field, which is being processed. In single query TEMP and BUFFTR will coincide at the beginning of every record.

After the checks have been cleared, the identification field has to be compared. Block IDENTF calls forth COMP. If the output MATCH = 1, then the process passes on to OTPT to output desired fields.

If the field does not match, the control branches off to NXREC which checks for record count and increments record counter. It adds record length to BUFFTR. Then it branches back to GTFD to start processing next record.

Transition from One Relation to Another

In a single query the transition is ea sy. There is no field to be passed on to next relation. Just select the next relation from TBPROC and start overlaying new infomation on the buffer.

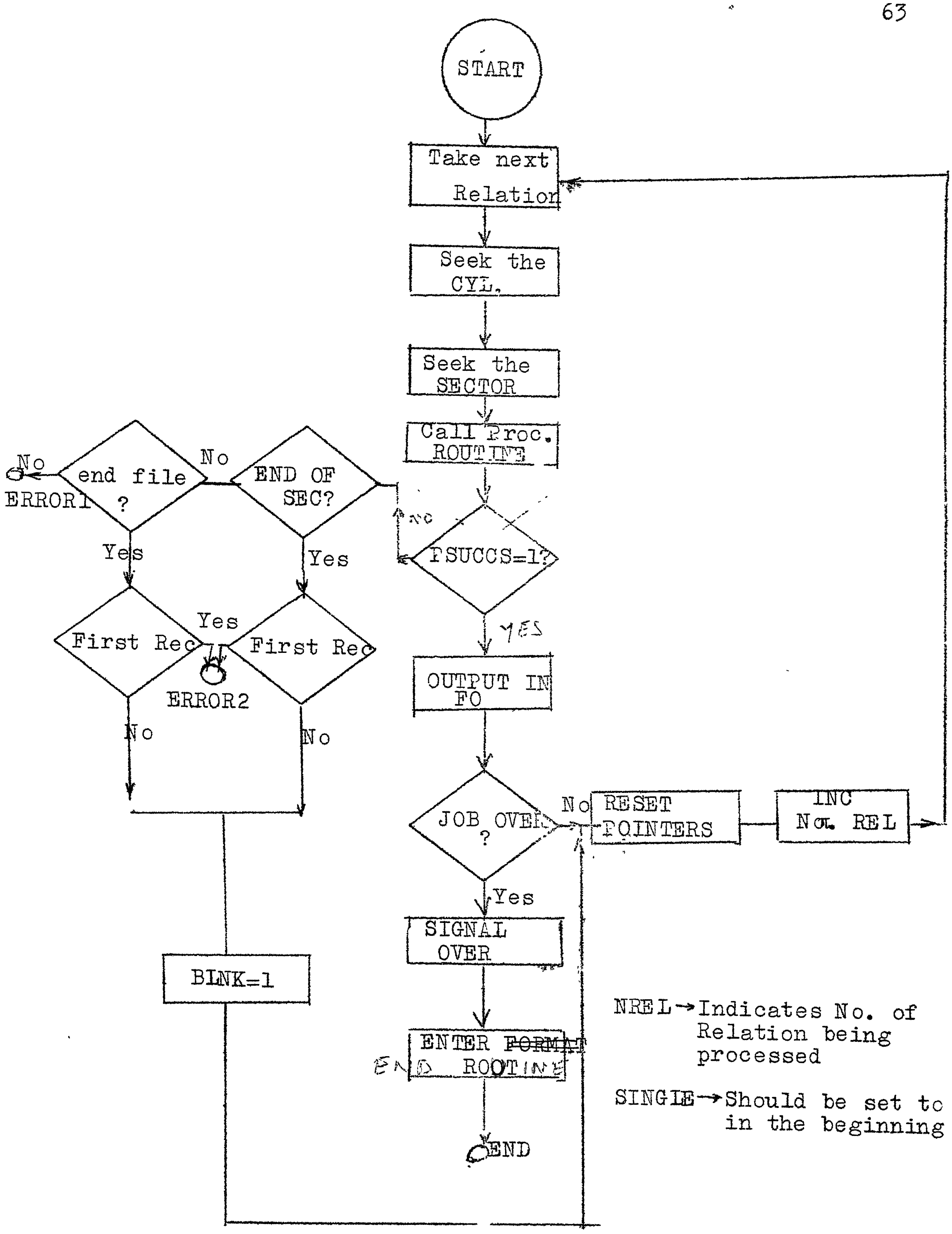
Processing End Conditions

It may happen that a particular relation infomation may not exist for that officer. For example, IC-20000, Capt. P. KUMAR may not be a graduate. As such his name will not figure in the relation QUAL. In this case, the seek mechanism will put a sector which should contain the information if it exists. An end of sector will be encountered without matching the key. The processing should not stop here but should go to next relation in TBPROC/TBOTPT and should fill blanks in this officer's qualification field. In the last sector End of File may similarly occur. The action will be same in this case. Only in first relation should the error be signalled in case of End Sector or End File Condition.

Imp ortant

The first relation in the query should be one containing all officers data.

-



NREL→Indicates No. of Relation being processed

SINGLE→Should be set to in the beginning

Figure 7-2. Flow Diagram of Single Query.

8. GROUP QUERIES

There may be two types of group queries:

(a) Queries about a group in which a subset of the complete relation is to be searched.

(b) Queries covering an entire relation.

<u>Subset Search</u>

There will be some condition on the primary key by which a particular block of information will be checked. The retrieval is in two phases -

(1) Seek the file portion which meets with the primary key condition.

(2) Process the relation. Keep a check on the primary key value and stop the job once the primary key value goes out of the range specified.

<u>Example</u>: Print out

Rank
Name
Unit

of <u>all</u> officers belong to corps of engineers.

In this case relation No. 1 will be CORPS and search will commence when key value equals engineers and stop when the key value changes.

<u>Example 2</u> Print out

Rank
Name
corps

of <u>all</u> officers whose personal number is less than IC-10000. The job gets over when the key value becomes IC-10,000.

<u>Full Relation Search</u>

This normally follows a search based not on primary key, search starts from the 1st sector of the relation and goes on till END of file has been reached.

<u>Example</u>

Print out the names of <u>all</u> officers whose <u>state</u> is <u>MAHARASHTRA</u> and whose <u>date of birth</u> is less than 01 January 1940.

<u>Differentiation Between Queries</u>

The first level of differentiation comes from putting a variable SINGLE. Now we use another variable GROUP. If GROUP = 1, it means the query is on primary key and a subset of a relation has to be searched. This condition has to be tested only after SINGLE has been tested. When both of them are zeroes it will mean a full relation search.

<u>Characteristics of a Group ENQUIRY</u>

The special characteristics of group queries are listed in the succeeding paragraphs.

1. <u>END OF JOB</u>

In a single query the job is over, once all the fields of OTIT table have been covered. The system then returns to initial conditions.

In a group inquiry, after the OTIT lable has been covered once, the system proceeds to next officers records.

It has to go back to the first relation and start processing from the point it last processed. End of job will come when

(a) A mismatch of primary key occurs or

(b) It reaches END OF FILE in relation No. 1.

These checks have to be made after every print out.

2. <u>No. of Pointers to be Kept</u>

Since the processing returns to the first relation after every successful record output, the pointer to the last record processed in Relation No. 1 has to be kept.

Also one pointer has to be maintained for every relation which is an inverted file or in which every record does not have a unique primary key.

3. <u>No. of Buffers Used</u>

In a single type query only one buffer is sufficient as no return to any relation is required. In a group query one buffer is required for the first relation in the TBPROC and one general buffer for other relations. In addition, one additional buffer will be required for each of the inverted fi relations, in which pointers have to be maintained.

4. <u>Transition from One Relation to Another</u>

In group query, the following information has to be kept alive

(a) Buffer for first relation

(b) Pointer in first relation to record last processed

(c) Buffers and pointers to inverted file relations.

The transition from one relation to another must not disturb any of the above. In addition, COND table entries have to be changed at every transition and old value remembered.

Example

We may be processing relation CORPS based on FD = 02, i.e., parent arms of the officer. The relation may look as given below:

FD	Corps 02	Personal No. 01	Name 03	Rank 04
	03	IC-10	PXR	Capt.
	03	IC-20	TCX	Lt.
	03	IC-25	ABC	Lt.
	03	IC-30	DYZ	2/Lt.

Once we have to search on FD02 = 03, i.e., Corps = Engineers, then the personal number condition becomes equal to ANY because all the above officers belong to Corps 03. The first relation will be brought into memory with a sector in which Corps = 03. Each officer will be taken one by one and his attributes in other relations found out. For other relations, it is like a SINGLE officer search every time one officer's record has to be processed. So when we are examining IC-20 TCX, the COND table will record PERSONAL NO. EQUALS IC-20 and PERSONAL NO. ANY once this particular officers attributes are over. The change over from ANY to EQUAL will be done after a successful hit in the first relation and will be changed back to ANY after the last relation in TBPROC has been processed.

5. <u>Action of Error Messages Come</u>

As in the case of SINGLE QUERY if an end of file comes in the first relation, the job is over. And end of file in the other relations means blanks have to be filled in those fields and proceed to next relation.

Every time a changeover of relation is taking place the action as described above has to be taken. In this case since personal number is the primary key of all relation except of Relation No. 1, the personal number of every officer can be put into KYFD and KFDLGTM. No change in COND FIELD will be required and all relations except for Relation No. 1 will take their personal number keys from KYFL instead of COND FIELD.

<u>Sequence of Processing Group Queries</u>

G1. <u>Initialize</u> : Set SINGLE = 0, NREL = 0, set pointer to beginning of buffers.

G2. Select the first relation.

G3. Seek the beginning of a group required in first relation, such a particular record in all other relations.

G4. Process the record against TBPROC.

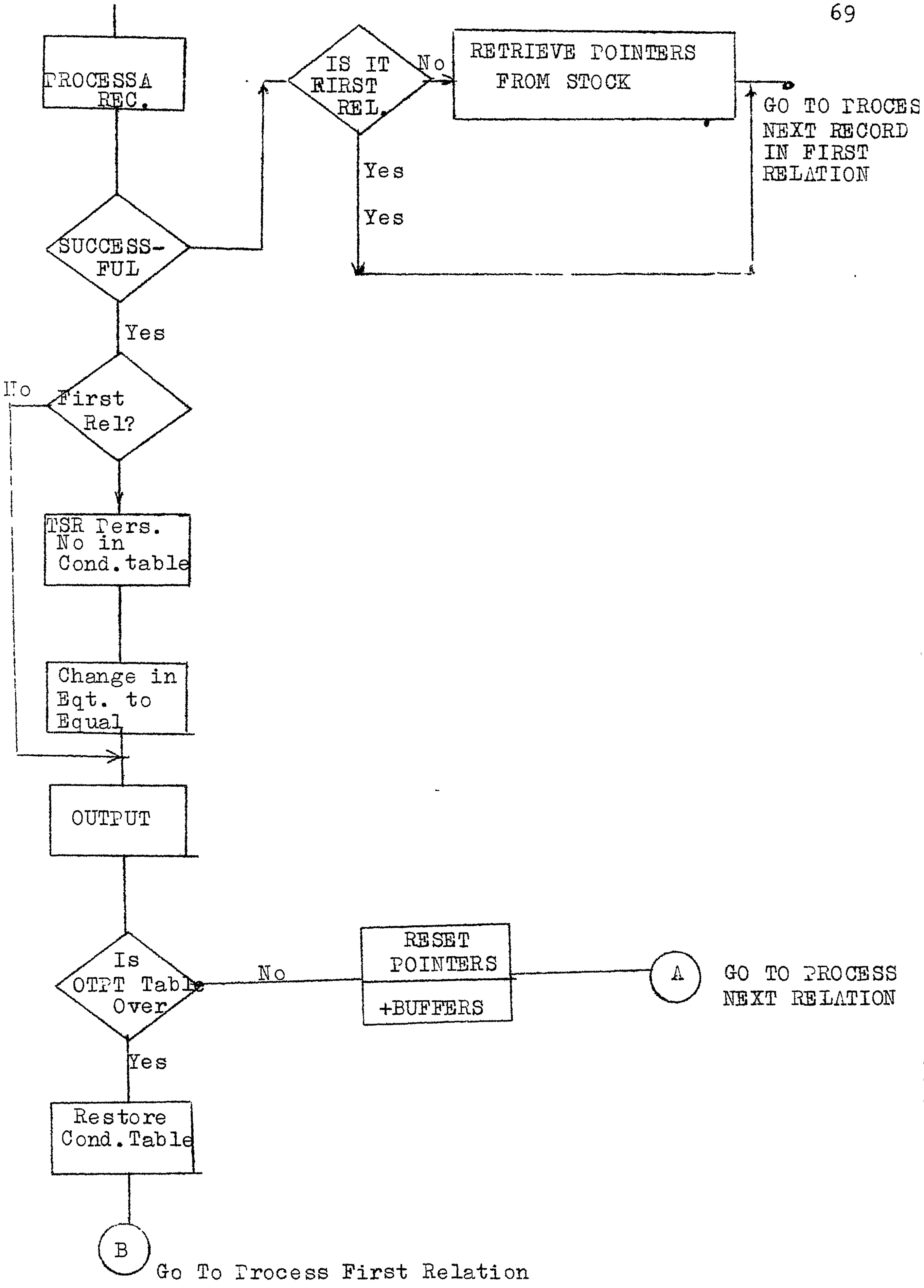
G5. If found OK put personal number in KYFD. Store the pointer in buffer in case of first relation.

G6. Rroceed to OUTPUT routines.

G7. Check whether all fields of this relation to be processed or over.

G8. If yes, save buffer and pointers settings, go to ~~6~~ G10 else go to next G9.

G9. Bring in next field relation. Go to G4.


PROCESSA REC.
SUCCESS-FUL
IS IT FIRST REL.
No
RETRIEVE POINTERS FROM STOCK
GO TO PROCES NEXT RECORD IN FIRST RELATION
Yes
Yes
Yes
No
First Rel?
TSR Pers. No in Cond.table
Change in Eqt. to Equal
OUTPUT
Is OTPT Table Over
No
RESET POINTERS
+BUFFERS
A
GO TO PROCESS NEXT RELATION
Yes
Restore Cond.Table
B
Go To Process First Relation

G10. Check whether all relations over. If yes Go To 12 else follow.

G11. If it is first relation, change buffer setting to BUFF2 change values in KYFD, KFDLGTM. Bring in new relation and go to G4. If it is not first relation nochange in settings required simply overlay.

G12. All relations over. Print OTPUT. Check whether query is over. ~~Go~~ If yes, GO TO G14, else G13.

G13. Change buffer settings to Buff 1. Go to process Relation No. 1 (G4).

G14. Go to End routine.

Action at the end of Sector Routine PROC F

Exit from Process

While processing a particular sector in a particular relation the following situations may arise.

(a) End of Sector.
(b) End of Relation.

In either case, there will be an exit from the PROC routine with a failure indicated by SWITCH2 = 0. The exit will also set another flag SWITCH3. SIWTCH3=0for end of sector and SWITCH3=1 for end of Relation. The action to be taken is as described below and is also indicated in the flow chart.

End of Sector

The end of sector action varies with the relation. It also depends upon whether it is a single query or a group query.

(a) First Relation

(i) If it is a single query end of sector means end of job.

(ii) If it is a group query it means action to go to next sector.

(b) Relation not First

End of sector signifies filling up the fields of the relation with blanks.

Incrementing Sector, Surface, Track

The following logic applies in incrementation.

1. Increment sector count. Check whether it exceeds 10. If so go to increment surface count and reset sector count to 1.
2. If the surface count exceeds 9, then go to next track from seek subroutines and primary index table.
3. If the track count exceeds number of tracks used as mentioned in Relation table mark END OF JOB.

END OF FILE

If it occurs in first Relation then it means end of job. In any other relation it will mean filling up blank for that relation and go to output routines.

Error

Exit from Proc. on any other account means error condition. Stop and give an error message.

Conclusion of Processing

Once the officers record has been processed and put in the output buffer, the routine FOTPT takes over and rearranges the output in the format of the subschema supplied by the user and control goes back to MAIN routine.

-

BLOCK AFTER PROCESS

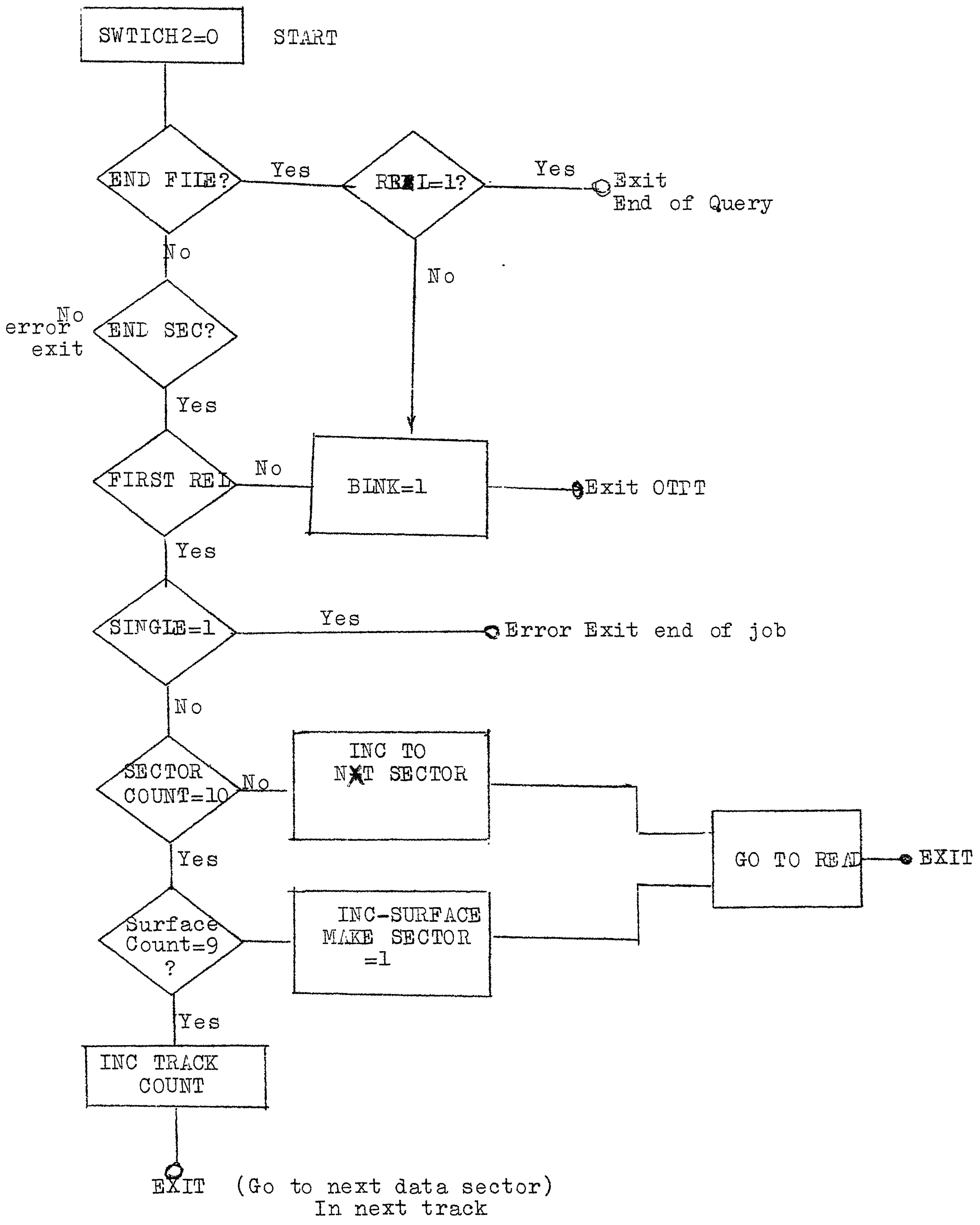


Figure 8-2. Flow Diagram Producer Procf.

9. OUTPUT ROUTINE

The processing routine, prepares a buffer for output in which the various fields are arranged in the order in which they appear in the table OTPT. This may not be the order in which the user requires the output. Thus another routine takes over which rearranges the order of the fields as per the requirement of the subschema.

Format

At the end of the process routine, the output buffer has the following shape.

FD IDENTITY	FD LENGTH	VALUE	MA RK	FD IDENTITY	FD LENGTH	VALUE
1 Byte	1 Byte	Variable	1 6			

Fig. 9-1. Layout of buffer at the end of processing.

The desired format will be as per subschema and will have the following shape.

FD VALUE 1	blanks	FD VALUE 2
xxxx		012

Figure 9-1a. Final output

Routine Working

Every time the OTPT routine prepares a line to be printed, it hands over to OUTPUT routine. The output routine makes use of the following data structures

(a) SBCHMA

It picks one field at a time from SBCHMA and scans the buffer for this field. On finding this field it puts the field length into a counter and transfers that many bytes to a temporary storage area.

Check Type of Field

It then reads the type of the field from SBCHMA. If it is numeric field, it has to be converted for printing. If it is alphanumeric it can be transferred straight. After conversion, this value is stored in the output buffer. Then the desired number of blanks are put in the output buffer. This information is also to be stored in the SBCHMA.

Printing

Once the output buffer is ready and the end of SBCHMA has been reached a command is given to the printer to print the desired information from output buffer. The whole line is printed and control goes back to process routine.

JOB OVER

Once the job is over, a message should come to the console and the printer. The printer skips a few lines and prints out a line of dashes to mark the end of table. The console prints out JOB OVER and the system returns to original state ready to accept new commands.

-

10. RECOMMENDATIONS

Due to the frequent failure of the TDC-316 system especially the input-output devices, it has not been possible to subject the system developed through rigorous tests. While working with the system, the following areas have shown need for improvement in design.

1. Query Translation

(a) At present the user needs to know the field identification for preparing fields. The subroutine operates on the numerical value of this field. It should be able to operate on the names of the fields given.

(b) There are three sets of cards to be fed in for making data structures

(i) TBPROC
(ii) COND
(iii) SUBSCHEMA

It is possible to reduce the number of sets to only two. TBPROC and COND can be combined. It will also aid in avoiding errors during processing arising out of differences while creating these data structures.

2. OUTPUT

As yet the system is geared for ~~out~~putting out ASCII coded information only. The final output routine FOTPT can be ~~xx~~ suitably modified with NOWR routine of DB routines to handle binary information to be output.

(c) <u>Data Structure Size</u>

The data structure sizes have been arbitrarily decided to meet normal operating conditions like

1. COND Table can handle 10 conditions

2. Max. field size for comparisons is 8 bytes

   so names cannot be a basis of search.

The sizes of the data structures can be modified to suit individual system requirement.

(d) <u>Inverted Files</u>

The system operates on one inverted file only. To incorporate more inverted files suitable modications will have to be made in the MAIN routine. Also, additional buffer of 256 bytes have to be reserved for each additional inverted file.

REFERENCES and BIBLIOGRAPHY

1. Martin, J., 'Principles of Data-Base Management', Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976, Chapter 1.

2. Engles, R.W., 'A Tutorial on Data Base Organization', Annual Review in Automatic Programming, Vol. 7, Part 1 (edited by Halpen and McGee), Pregamon Press, July 1972.

3. Fry, J.P., and Sibley, E.H., 'Evolution of Data Base Management Systems', Computing Survey, Vol. 8, No. 1, March 1976.

4. Martin, J., 'Computer Data Base Organization', Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975.

5. Date, C.J., 'An Introduction to Data Base Systems,' Addison-Wesley Publishing Company, Inc., Phillipines, 1975.

6. Haseman, W.D., et. al., 'A Partial Implementation of CODASYL DBTG Report as an Extension to FORTRAN', Management Datamatic, Vol. 4, No. 5, October 1975.

7. Tsichritzis, D.C., and Lochinsky, P.H., 'Hierarchical Data-Base Management : A Survey', Computing Survey, Vol. 8, No. 1, March 1976, pp. 105-123.

Codasyl Data Base Task Group, April 1971 Report, ACM, New York.

Codd, E.F., 'A Relational Model of Data for Large Shared Data Banks', Communication of ACM, Vol. 13, No. 6, June 1970.

'hamberlin, Donald, D., 'Relational Data-Base Management stems', Computing Survey, Vol. 8, No. 1, March 1976, pp. 56.

APPENDIX 1: Method of Using the System

The information retrieval system occupies memory space from 60,000 to 72,000 (octal address). The routine approach starts at 72000. The system makes use of the following external routines:

(a) SCAN called by letter SC line feed.

(b) Read

(c) SET.

Calling Instructions and Dialogue

The routines are placed in the system area of the disk and are placed with KDM. The procedure is as follows:

1. Load KDM
2. Start from 0173500. An asterisk will be typed on the teletype.
3. Type 'SC' line feed. An asterisk will again be typed.
4. Type 'AP' lf. Approach will be loaded.
5. Type 'MN' lf. Main will be loaded.
6. Set address 72000 from console and press start. It will start the dialogue.

Dialogue

The starting message will be

GIVE ID NO.

Fill in the ID No. for that day (at present 123) and press LF. If the number is incorrect, the teleprinter will type a message.

'UNAUTHORIZED USER' and start ringing the bells. Start again from 72000.

If the ID is correct, the following message will be typed:

```
USER VERIFIED PROCEED .
FOR UPDATE TYPE IN U
FOR RETRIEVE TYPE IN R
IF DBA TYPE IN D
```

Press R LF then the system types out

```
STARTING RETRIEVAL, INDICATE TYPE OF QUERY
IF SINGLE OFFICER QUERY, PRESS S, ELSE G LF
```

Here the system finds out the type of query from the user. If proceeds differently in both cases. In case of single query, the only entry in condition table will be his personal number.

<u>Action on Single Query</u>

If you press S LF the system will type out

```
SINGLE QUERY, IF INPUT DEVICE KARD PRESS 'C' LF.
IF INPUT DEVICE KEY B PRESS KF LF.
```

Depending upon the type of input device press C or K LF.

The machine then types out

```
TYPE IN LEVEL FDNAME, FDID, JOB ON THE FD.
```

<u>Definition for FIELDS</u>

The user has to now give the list of fields which he wants to process or output the format of this information is

Level	02
FDNAME	Any 6 characters. First character to be an alpbabet.
FDID	FD No. 1st letter should be F e.g., F7 or F07.
JOB	Blank for 0, 9 for putting 1 in TBPROC, 99 for putting 2. (0 means output only, 1 process and output, 2 process only)

Put a blank between each entry and end with a (.) LF. e.g.,

02 ƀ NAME ƀ F01 ƀ 9 ƀ . LF.

The system prints out the message e.g., field defination and asks for NEXT FD. Type in all the fields and finally end the field listing with (.) LF which will put zeroes in the last entry in the table for processing and output and print out

ALL FIELDS OVER PRINTING TABLE FOR PROCESSING AND OUTPUT.

Then are the table is printed out. It consists of ten lines each of five entries.

The columns are as follows:

1st	Relation No.
2nd	FD No.
3rd	Field length
4th	Distance of field from beginning of record
5th	Job to be done on this field.

After it prints out the TBPROC, it prints a message

TYPE IN PERSONAL NO.

Fill in the personal number and press LF. The system registers this personal number and then gives message

COND TABLE OVER GIVE SUBSCHEMA.

It also asks for choice of INPUT DEVICE, keyboard or Card output. Now give the list of fields in the order in which the user desires them to be output, e.g.,

F1
F5
F6
F4

The definition of the fields will be in the same pattern as described earlier. Every time a field has been entered, the bell rings for a new field. End the subschema with a (.) LF. The system then types

```
SUBSCHEMA OVER
JOB OVER
PRESS CONTINUE TO START RETRIEVAL.
```

Press continue to start retrieval of information.

Group Query

When you press G in response to type of query it finds

```
GROUP INQUIRY  GIVE FD ON WHICH SEARCH IS BASED.
```

In response to this message type in the field specifications on which the search is based. It is primarily meant to use for retrieval of inverted file Relation 1, which is based on FD 03. The system types out the details of field and asks for choice of INPUT device. Then the dialogue is same as for SINGLE query till the printing of TBPROC is over. Then to build condition table it again asks for choice of INPUT device. Once the input device is mentioned it asks for

```
GIVE CONDITION FD
```

Specify the FD No. as done earlier. The system registers it and asks

```
GET VALUE.  AFTER VALUE GIVE ONE BLANK LF
```

Type in value and press LF. The value is registered and the system asks

```
GIVE CONDITION.
```

Type in the condition for this field may be

```
EQUAL
NEQ
GT
GE
LT
LE
ANY   and press LF.
```

The system registers this condition and goes back to ask

GIVE CONDITION FD.

The user fills up all the condition entries and then as the last entry enters (.) LF.

The condition table is over and now the system goes to make subschema. The dialogue is same as for single query. In case the user selects the card as input, all the condition field entries are put on the card. See example below.

<u>CARD INPUT FOR THE SYSTEM</u>

There are three series of card inputs. These are in the order and format given below.

For TBPROC

```
02 ƀ XYS ƀ F1 ƀ 9/99/ƀ ƀ . LF
:
0 ɪ LF - (last card)
```

<u>For COND Table</u>

```
02 ƀ XYZ ƀ FN' ƀ ƀ ƀ . VAL ƀ COND
:
02   ABC   F2        . 100   EQUAL
:
0 .  LF (last card)
```

For SUBSCHEMA

Since these are only output cards

```
02 ƀ NAME ƀ FN ƀ ƀ ƀ . LF
:
0 . LF
```

cards for all the groups can be kept on the card reader in one stack.

APPENDIX 2. Programme Layout

The programme is divided into the following parts with starting locations as shown:

		Location
1. MAIN		60,000
2. SEEK Routines		
	(a) FNDKEY	62,000
	(b) FNDIX	62,400
	(c) TRKDS	63,000
	(d) FNCSEC	63,300
3. Processing Routines		
	(a) Proc.	64,000
	(b) Proc F	64,600
4. OUTPUT Routines		
	(a) OTPT	65,500
	(b) FOTPT	66,200
5. Query Translation		
	(a) Approach	72,000
6. GEN ROUTINE COMP		61,332
7. GEN. VARIABLES		67,300

These routines have already been described in the various chapters. MAIN routine calls the other subroutines turn by turn for retrieval.

In addition it makes use of external routines from DB. These are

(a) SCAN
(b) GET
(c) READ
(d) SET.

These routines are loaded from 20,000. After calling KDM. Load DB.

LISTING

The listings size does not permit it to be attached to this thesis. Copies of listings are kept in the computer centre library.

Functions

The main routine is the controlling routines. The SEEK routines locate the relevant sector of information on the disk and put it in memory buffer. The processing routines then process the information in the buffer. The output routines prepare output buffers as per the subschema given by the user and output it.

The query translation elicits specifications from the user and prepares data structures, COND TABLE, Processing table and Subschema.

APPENDIX 3. Working with TDC - 316

This project was taken up on TDC-316 especially to try out the TDC-316 system with an intensive usage. The project has been successful in this respect as it has subjected the TDC-316 to real heavy loading. The system at the Indian Institute of Technology, Kanpur has proved to be very unreliable with a very heavy down time. The main points observed are listed below:

1. CARD READER	:	Break down in blower system, too frequent pick checks, resulting in heavy loss of input time.
2. PRINTER	:	An old model printer which gave extensive trouble in paper feed mechanism for over a month and a half.
3. DISK7	:	No protection of system programs possible. Disk heads got damaged once. Disks scratch very early.
4. KEYBOARD	:	Off line OK. ON LINE possibly due to bad connections at CPU gives very frequent troubles. Also the model is light duty model, should not be used more than 2 hours at a strach serious time taken
5. PAPER TAPE SYSTEM	:	By and large satisfactory.
6. CONSOLE	:	Almost trouble free. Power supply gives trouble.
7. CPU	:	Frequent troubles in disc controller. Hardware bootstrap and keyboard cards.
8. MEMORY	:	Very good. Except for failure in power supplies it has been trouble free.

Due to frequent failures of the PERIPHERALS, the system was almost totally off from April to mid-June 1977. Even when all other subsystems are working fine, the inefficiency of the card reader increases the assembly time/compute time very badly.

Software

The KDM worked satisfactorily. BAL has also proved efficient in assembly and execution. Assembly Language Programs use very small memory space. COPY routines did not work while creating an object tape from DISK.

Due to lack of protection on the disc and inefficiency of the card reader, it is strongly recommended to users to create object program paper tapes during assembly for future input.